Lab 3

Name : Anish S Ghiya

Registration Number : 18BCB0002

Task 1

- **Import dataset from sklearn**
- **Make the dataframe from the data**
- **Describe and info on dataset**
- **Value_Counts for target**
- **Groupby on target**
- **Train test split using sklearn (stratify(with and without) & randomstate)**
- **Binarisation using pd.cut**
- **Predictions based on threshold of b**
- **Calculation accuracy**

Importing Libraries

In [ ]:

```
import sklearn.datasets
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

Loading the dataset

Breast cancer dataset from the sklearn library

In [ ]:

```
cancer_ds = sklearn.datasets.load_breast_cancer()
X = cancer_ds.data
y = cancer_ds.target
y_names = cancer_ds.target_names
print(y[:10])
print(y_names)
print(X[:1])
```

```
[0 0 0 0 0 0 0 0 0 0]
['malignant' 'benign']
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]]
```

Generating the dataframe for the dataset

In [ ]:

```
cancer_df = pd.DataFrame(cancer_ds.data, columns=cancer_ds.feature_names)
cancer_df['target'] = y
cancer_df['target_name'] = cancer_df['target'].map({0:y_names[1], 1:y_names[0]})
cancer_df.head()
```

Out[ ]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	pe
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	1.0950	0.9053	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.7456	0.7869	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0.4956	1.1560	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	0.7572	0.7813	

Getting the general description of the dataset

Using the describe and info methods

In [ ]:

```
cancer_df.describe()
```

Out[ ]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	f dime
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.0
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.0
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.0
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.0
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.0
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.0
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.0
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.0

In [ ]:

```
cancer_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
```

```
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64
 31  target_name              569 non-null    object
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

Getting the target value description

Using the value_counts function to get the rows of each value in the target variable

In [ ]:

```
display(cancer_df['target_name'].value_counts())
display(cancer_df['target'].value_counts())
```

```
malignant    357
benign       212
Name: target_name, dtype: int64
```

```
1    357
0    212
Name: target, dtype: int64
```

Finding mean of all columns for specified target

Using the groupby function to calculate mean of each of the target variable values

In [ ]:

```
cancer_df.groupby(['target', 'target_name'],as_index=False).mean()
# observation from output the values for the target "0"(benign) is more than that of targe
t "1" (malignant)
```

Out[ ]:

	target	target_name	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mea symmet
0	0	benign	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775	0.087990	0.1929
1	1	malignant	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058	0.025717	0.1741

Splitting dataset into train and test

Without stratify and with stratify

Random_state is used to maintain consistency throughout all the splits we generate

In [ ]:

```
X=cancer_df.drop(['target'],axis=1)
y=cancer_df['target']

X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=0.1,
                                                    random_state=42,
                                                    )
```

In [ ]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(512, 31)
(512,)
(57, 31)
(57,)
```

In [ ]:

```
print(y_train.value_counts())
print(y_test.value_counts())
```

```
1    317
0    195
Name: target, dtype: int64
1    40
0    17
Name: target, dtype: int64
```

In [ ]:

```
print(y_train.value_counts()[1]/ y_train.value_counts()[0])
print(y_test.value_counts()[1]/y_test.value_counts()[0])
```

```
1.6256410256410256
2.3529411764705883
```

The distributuion without stratified is skewed towards malignant

Hence we will be using the stratify parameter to reduce this skew in the train test split

In [ ]:

```
from sklearn.model_selection import train_test_split

X=cancer_df.drop(['target','target_name'],axis=1)
y=cancer_df['target']

X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=0.1,
                                                    random_state=42,
                                                    stratify=y)
print(y_train.value_counts())
print(y_test.value_counts())
print("\n")
print(y_train.value_counts()[1]/ y_train.value_counts()[0])
print(y_test.value_counts()[1]/y_test.value_counts()[0])
```

```
1    321
0    191
Name: target, dtype: int64
1    36
0    21
Name: target, dtype: int64


1.680628272251309
1.7142857142857142
```

From this we can clearly note the reduction in the skew as the disributions are fairly similar

In [ ]:

```
X_train
```

Out[ ]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error
413	14.990	22.11	97.53	693.7	0.08515	0.10250	0.06859	0.03876	0.1944	0.05913	0.3186	1.3360
517	19.890	20.26	130.50	1214.0	0.10370	0.13100	0.14110	0.09431	0.1802	0.06188	0.5079	0.8737
245	10.480	19.86	66.72	337.7	0.10700	0.05971	0.04831	0.03070	0.1737	0.06440	0.3719	2.6120
102	12.180	20.52	77.22	458.7	0.08013	0.04038	0.02383	0.01770	0.1739	0.05677	0.1924	1.5710
28	15.300	25.27	102.40	732.4	0.10820	0.16970	0.16830	0.08751	0.1926	0.06540	0.4390	1.0120
...	...	...	...	...	...	...	...	...	...	...	...	...
509	15.460	23.95	103.80	731.3	0.11830	0.18700	0.20300	0.08520	0.1807	0.07083	0.3331	1.9610
230	17.050	19.08	113.40	895.0	0.11410	0.15720	0.19100	0.10900	0.2131	0.06325	0.2959	0.6790
354	11.140	14.07	71.24	384.6	0.07274	0.06064	0.04505	0.01471	0.1690	0.06083	0.4222	0.8092
103	9.876	19.40	63.95	298.3	0.10050	0.09697	0.06154	0.03029	0.1945	0.06322	0.1803	1.2220
248	10.650	25.22	68.01	347.0	0.09657	0.07234	0.02379	0.01615	0.1897	0.06329	0.2497	1.4930

512 rows × 30 columns

Binarising the dataset

Using the pd.cut option with the bins to be 2

In [ ]:

```
X_binary_tr = X_train.apply(pd.cut, bins=2, labels=[1,0])
X_binary_te = X_test.apply(pd.cut, bins=2, labels=[1,0])
X_binary_tr.head()
```

Out[ ]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error
413	1	1	1	1	1	1	1	1	1	1	1	1
517	0	1	0	1	0	1	1	1	1	1	1	1
245	1	1	1	1	0	1	1	1	1	1	1	1
102	1	1	1	1	1	1	1	1	1	1	1	1
28	1	0	1	1	0	1	1	1	1	1	1	1

In [ ]:

```
X_binary_te.head()
```

Out[ ]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error
199	1	1	1	1	1	1	1	1	1	1	1	1
288	1	1	1	1	1	1	1	1	0	1	1	0
73	1	1	1	1	1	1	1	1	1	1	1	1
121	0	1	0	1	1	1	1	1	1	1	0	0
14	1	0	1	1	0	0	0	1	0	0	1	1

In [ ]:

```
X_binary_tr = X_binary_tr.values
X_binary_te = X_binary_te.values
```

Simple ML model

Using the value of b to be the threshold, the values are predicted for the training dataset

Based on the number of correct and wrong predictions the accuracy is calculated

Formula

> ***Correct predictions/(total number of predictions)***

In [ ]:

```
b = 20
```

In [ ]:

```
def get_predictions(X_binary_tr, y_train, b):
  pred = []

  # loop to go through the all the values of b for all columns
  for i in range(X_binary_tr.shape[0]):
    # for checking the threshold for predicting the bening and makignant
    if (np.sum(X_binary_tr[i,:]) >= b):
      pred.append(1)
    else :
      pred.append(0)

  # creating a dataframe with predictions and true values
  df = pd.DataFrame(columns=['Predictions', 'True_values'])
  df['Predictions'] = pred
  df['True_values'] = y_train

  #checking if the prediction is sane as y_train
  df['cor_wr'] = df['Predictions']==df['True_values']
  display(df['cor_wr'].value_counts())

  #accuracy = correct_preds/(correct_preds + wrong_preds)
  return df['cor_wr'].value_counts()[1] / (df['cor_wr'].value_counts()[1]+df['cor_wr'].va
lue_counts()[0])


# Iterating through all possible values of b
for b in range(X_train.shape[1]):
  print("Value of b: ", b)
  print("Accuracy: ", get_predictions(X_binary_tr, y_train, b))
```

```
Value of b:  0

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  1

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  2

True     285
False    227
```

```
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  3

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  4

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  5

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  6

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  7

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  8

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  9

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  10

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  11

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  12

True     285
False    227
Name: cor_wr, dtype: int64

Accuracy:  0.556640625
Value of b:  13

True     284
False    228
```

```
Name: cor_wr, dtype: int64

Accuracy:  0.5546875
Value of b:  14

True     284
False    228
Name: cor_wr, dtype: int64

Accuracy:  0.5546875
Value of b:  15

True     281
False    231
Name: cor_wr, dtype: int64

Accuracy:  0.548828125
Value of b:  16

True     281
False    231
Name: cor_wr, dtype: int64

Accuracy:  0.548828125
Value of b:  17

True     279
False    233
Name: cor_wr, dtype: int64

Accuracy:  0.544921875
Value of b:  18

True     278
False    234
Name: cor_wr, dtype: int64

Accuracy:  0.54296875
Value of b:  19

True     277
False    235
Name: cor_wr, dtype: int64

Accuracy:  0.541015625
Value of b:  20

True     276
False    236
Name: cor_wr, dtype: int64

Accuracy:  0.5390625
Value of b:  21

True     273
False    239
Name: cor_wr, dtype: int64

Accuracy:  0.533203125
Value of b:  22

True     271
False    241
Name: cor_wr, dtype: int64

Accuracy:  0.529296875
Value of b:  23

True     278
False    234
Name: cor_wr, dtype: int64

Accuracy:  0.54296875
Value of b:  24

True     271
False    241
```

```
Name: cor_wr, dtype: int64

Accuracy:  0.529296875
Value of b:  25

True     265
False    247
Name: cor_wr, dtype: int64

Accuracy:  0.517578125
Value of b:  26

True     259
False    253
Name: cor_wr, dtype: int64

Accuracy:  0.505859375
Value of b:  27

False    265
True     247
Name: cor_wr, dtype: int64

Accuracy:  0.482421875
Value of b:  28

False    271
True     241
Name: cor_wr, dtype: int64

Accuracy:  0.470703125
Value of b:  29

False    281
True     231
Name: cor_wr, dtype: int64

Accuracy:  0.451171875
```

In [ ]:

```

```